Rose-Hulman Undergraduate Mathematics Journal

Volume 12
Issue 1

Article 5

A Mathematical Analysis of The Generalized Oval Track Puzzle

Samuel Kaufmann
Carnegie Mellon University, sakaufma@andrew.cmu.edu

Rose-
Hulman
Undergraduate
Mathematics
Journal

A Mathematical Analysis of The Generalized Oval Track Puzzle

Samuel Kaufmann[a]

Volume 12, no. 1, Spring 2011

Sponsored by

Rose-Hulman Institute of Technology
Department of Mathematics
Terre Haute, IN 47803
Email: mathjournal@rose-hulman.edu
http://www.rose-hulman.edu/mathjournal

[a]Carnegie Mellon University, sakaufma@andrew.cmu.edu. Research Advisor: Dr. Richard Statman, Department of Mathematics ⋄ Carnegie Mellon University

Rose-Hulman Undergraduate Mathematics Journal
Volume 12, no. 1, Spring 2011

A Mathematical Analysis of The Generalized Oval Track Puzzle

Samuel Kaufmann

Abstract. The oval track puzzle (also known as Top Spin) is a game consisting of 20 numbered tiles in an oval shaped track. Also, there is a fixed window (the swapping window) of 4 tiles that reverses the order of the tiles within the window, leaving the other 16 tiles fixed. The object of the puzzle is to reorder the tiles into counting order using the mechanisms of the puzzle. Previously, conditions for both solvability and non-solvability for the generalized oval track puzzle with n total tiles and k tiles in the swapping window were shown. We will now prove tight asymptotic bounds on the number of swaps needed to solve any configuration of a puzzle with n total tiles and k tiles in the swapping window provided that n and k yield a solvable case to begin with. These bounds will be asymptotic because we will assume that n grows infinitely and k stays fixed.

Introduction

This paper explores the mathematical properties of the generalized version of the oval track puzzle *Top Spin*. *Top Spin* is a puzzle consisting of 20 tiles numbered from 1 to 20 in an oval shaped track. There is also a fixed window (the *swapping window*) of 4 tiles that reverses the order of the tiles within the window, leaving the other 16 tiles fixed. The object of the puzzle is to reorder the tiles into counting order using the mechanisms of the puzzle (i.e. sliding the tiles in and out of the swapping window and reversing the order of the tiles in the swapping window). The generalized oval track puzzle is such a puzzle with n tiles and a swapping window of size k.

When researching this puzzle, the first question that came up was, "What values of n and k would yield a solvable puzzle?" This question was partially answered by E. Wilbur [3], but many cases were left as open questions. The first section of this paper builds on the work that Wilbur did by reproving his solved cases as well as solving the cases he left open.

The next question that came up was, "Given a solvable puzzle, how many puzzle operations would it take to solve it?" The proofs of two of the solvable cases in the first section yielded algorithms for generating permutations that could be used to solve a puzzle in those cases. By assuming n and k yield a solvable puzzle, and by letting n grow infinitely as we fix k, we were able to design an algorithm to solve a puzzle using said permutations. This provided an asymptotic upper bound of $O(n^2)$ which we were then able to make into a tight asymptotic bound by providing an equivalent lower bound of $\Omega(n^2)$. The second section of this paper describes the algorithm and proves the lower bound and consequently the tight bound.

Note that the first section of this paper was published as an article in an edition of Rose-Hulman's Undergraduate Mathematics Journal. The first paper [1] was published by myself and Andreas Kavountzis. Although Wilbur's paper provided helpful guidance in the research used to write the first paper, most of the help I received on the material for that paper came from my research partner at the time, Andreas Kavountzis. I dedicate this work in part to Mr. Wilbur, Mr. Kavountzis, Dr. Richard Statman (my research advisor), as well as my family, friends, and professors at Carnegie Mellon University.

1 Proof of Solvability

1.1 Terminology

To facilitate easy demonstration of game movements we shall employ a linear notation (i.e. a string of numbers) that begins with the left-most member of the *swapping window* in considering permutations. Here we tacitly use the notion of *position* when determining permutations by using this notation. However, at times, mostly for the purpose of illustration, we will not fix the *location* of the swap window on the string. Finally, elements in the swap window will be enclosed by parentheses.

Definition: *We represent the puzzle consisting of n total tiles and k tiles in the swapping window by the 2-tuple (n, k).*

Remark: We assume that the bounds $2 \leq k < n$ must hold for the mechanisms of the puzzle to work.

Definition: *The puzzle (n, k) is solved if the puzzle's configuration is in the identity configuration (i.e. 1 2 ... n).*

Definition: *A swap is the operation defined by the reflection of the k tiles in the swapping window. Since the value of k can be odd or even, we have two cases for the way in which the k tiles in the swapping window can be rearranged:*

- **Even Case:**
  Let $a_1, a_2, \ldots, a_k$ be the k tiles inside the swapping window, in the order shown, with a_1 in position 1. For k even, we have that one swap will yield a new permutation equal to $(a_1 a_k)(a_2 a_{k-1}) \ldots (a_{\frac{k}{2}} a_{\frac{k}{2}+1})$.

- **Odd Case:**
  Let $a_1, a_2, \ldots, a_k$ be the k tiles inside the swapping window, in the order shown, with a_1 in position 1. For k odd, we have that one swap will yield a new permutation equal to $(a_1 a_k)(a_2 a_{k-1}) \ldots (a_{\lceil \frac{k}{2} \rceil})$.

Definition: *A (clockwise) translation is the movement defined by sliding the tiles within the track such that any tile in position i is moved to position i + 1 for i = 1, ..., n - 1. In other words, if we let 1, 2, ..., n indicate the positions of the tiles in the track, then a translation can be described by the n-cycle (1 2 ... n).*

Remark: Note that applying a swap *twice* or applying a translation n times both leave the puzzle unchanged. Furthermore, due to the circular nature of the puzzle, we consider permutations to be equivalent up to translation.

1.2 Solvability

Before discussing the solvability of the puzzle, we need to give the following definitions:

Definition: *The family of all permutations of {1, 2, ..., n} is called the symmetric group on n-letters, denoted by S_n* [4, pg. 107].

Remark: *Any permutation $\pi \in S_n$ is either a cycle or can be factored into a product of disjoint transpositions* [4, pg. 113].

Definition: *The subset of S_n consisting of all even permutations is called the alternating group, denoted by A_n* [4, pg. 150].

Throughout this paper, we will be presenting cases for which the puzzle is solvable and cases for which the puzzle is not solvable. However, what does it mean for the puzzle to be solvable? The following theorem addresses this question.

Theorem 0: *The puzzle (n, k) is solvable if and only if we can generate all permutations $\pi \in S_n$ using the mechanisms of the puzzle.*

Proof:
1. If we can generate all permutations $\pi \in S_n$, then the puzzle (n, k) is solvable.

Assume we can generate all permutations in S_n. If this were the case, then we could go from any puzzle configuration x to the identity configuration by applying a permutation. This permutation is characterized by undoing the moves that it took to get to x. Thus, we can solve the puzzle.

2. If the puzzle (n, k) is solvable, then we can generate all permutations $\pi \in S_n$.

If the puzzle is solvable, then we have to be able to switch adjacent tiles. For instance, if the configuration was **1 3 2 4 . . . n**, then we would have to switch tiles 2 and 3 in order for the puzzle to be solved. Thus, if we can switch adjacent elements, we can perform disjoint transpositions, and since any permutation $\pi \in S_n$ is the product of disjoint transpositions, we can generate all permutations in S_n.✓

Thus, for any puzzle *(n, k)*, we will prove that it is solvable by generating S_n. To prove it is unsolvable, we will show that it is not possible to generate S_n. We now examine the possible cases for values of n and k.

Theorem 1: *If $n \equiv 0$ (mod 2) and $k \equiv 0$ (mod 4) or $k \equiv 2$ (mod 4), then the puzzle is solvable.*

Proof:
To prove this, we will present an algorithm that switches adjacent tiles in the puzzle. If we can create an adjacent transposition, then we can go from any configuration to the identity configuration by simply transposing the elements pairwise into their correct positions. The algorithm uses a special move called a *swap-translation*. Like the name describes, a swap-translation is a swap immediately followed by a translation. Note that when using a swap-translation, one needs to specify which direction the translation is going in. Using this special

move, we can go through the steps of the algorithm. Assume without loss of generality that the starting configuration of the puzzle is the identity configuration

$$\mathbf{(1\ 2\ \ldots k)\ k{+}1\ \ldots n}$$

Our goal is to swap the elements 1 and 2, fixing all other elements, so that our final configuration after applying the algorithm is

$$\mathbf{(2\ 1\ \ldots k)\ k{+}1\ \ldots n}$$

Now we start the algorithm:

(1) First, we fix 1 and 2 to the left of the swapping window, making the configuration

$$\mathbf{1\ 2\ (3\ \ldots k\ k{+}1\ k{+}2)\ k{+}3\ \ldots n}$$

(2) Next, we fix the rightmost $k-1$ elements of the swap window and perform $n-(k-1)-2 = n-k-1$ counterclockwise swap-translations. This will bring us to the configuration

$$\mathbf{(k{+}1\ k\ \ldots 1)\ 2\ \ldots n}$$

(3) Now we perform $\frac{k}{2}+1$ swap-translations, alternating the direction we translate, beginning with a counterclockwise translation. This step moves 1 and 2 into the two center positions of the swap window. This gives us the configuration

$$\mathbf{(\ldots 2\ 1\ \ldots)\ \ldots n\ k}$$

if $k \equiv 0 \pmod 4$ and the following configuration

$$\mathbf{(\ldots 1\ 2\ \ldots)\ \ldots n}$$

if $k \equiv 2 \pmod 4$.

(4) Now we perform $\frac{k}{2}$ swap-translations, alternating the direction we translate, beginning with a clockwise translation if $k \equiv 0 \pmod 4$ and a counterclockwise translation if $k \equiv 2 \pmod 4$. This brings us to the configuration

$$\mathbf{2(1\ \ldots k\ k{+}1)\ k{+}2\ \ldots n}$$

which, if we perform one more clockwise translation, gives us

(2 1 . . . k) k+1 . . . n

An example of this algorithm is given in Figure 1.1 below for both possible values of k. Given this algorithm, we can solve any puzzle in the case of $n \equiv 0 \pmod 2$ and $k \equiv 0 \pmod 4$ or $k \equiv 2 \pmod 4$.✓

$$\begin{array}{c|c}
\left(\underline{1}\ \underline{2}\ \underline{3}\ \underline{4}\right)\ \underline{5}\ \underline{6}\ \underline{7}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12} & \left(\underline{1}\ \underline{2}\ \underline{3}\ \underline{4}\ \underline{5}\ \underline{6}\right)\ \underline{7}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12} \\
\downarrow_{Step\ 1} & \downarrow_{Step\ 1} \\
\underline{1}\ \underline{2}\ \left(\underline{3}\ \underline{4}\ \underline{5}\ \underline{6}\right)\ \underline{7}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12} & \underline{1}\ \underline{2}\ \left(\underline{3}\ \underline{4}\ \underline{5}\ \underline{6}\ \underline{7}\ \underline{8}\right)\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12} \\
\downarrow_{Step\ 2} & \downarrow_{Step\ 2} \\
\left(\underline{5}\ \underline{4}\ \underline{3}\ \underline{1}\right)\ \underline{2}\ \underline{6}\ \underline{7}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12} & \left(\underline{7}\ \underline{6}\ \underline{5}\ \underline{4}\ \underline{3}\ \underline{1}\right)\ \underline{2}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12} \\
\downarrow_{Step\ 3} & \downarrow_{Step\ 3} \\
\left(\underline{5}\ \underline{2}\ \underline{1}\ \underline{3}\right)\ \underline{6}\ \underline{7}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12}\ \underline{4} & \left(\underline{4}\ \underline{3}\ \underline{1}\ \underline{2}\ \underline{7}\ \underline{6}\right)\ \underline{5}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12} \\
\downarrow_{Step\ 4} & \downarrow_{Step\ 4} \\
\left(\underline{2}\ \underline{1}\ \underline{3}\ \underline{4}\right)\ \underline{5}\ \underline{6}\ \underline{7}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12} & \left(\underline{2}\ \underline{1}\ \underline{3}\ \underline{4}\ \underline{5}\ \underline{6}\right)\ \underline{7}\ \underline{8}\ \underline{9}\ \underline{10}\ \underline{11}\ \underline{12}
\end{array}$$

Figure 1: An example of the adjacent transposition algorithm in the case of $n = 12,\ k = 4$ (on the left) and $n = 12,\ k = 6$ (on the right).

Theorem 2: *If $n \equiv 1$ (mod 2) and $k \equiv 3$ (mod 4), then the puzzle is solvable.*

Proof:
We show this case by first showing the existence of a $k - cycle$ which we will use to find a $3 - cycle$. Define τ to be a clockwise translation, and σ to be a swap. Now applying the following sequence of moves to the puzzle (read from left to right) we will arrive at a $k - cycle$.

$$(\tau\sigma)^{n-k}\tau = (\tau\sigma\tau\sigma \ldots \tau\sigma)\tau$$

Written linearly as:

(1 . . . k) k+1 . . . n

$$\begin{array}{c}
\mathbf{(n\ 1\ \ldots k\text{-}1)\ k\ \ldots n\text{-}1} \\
\mathbf{(k\text{-}1\ \ldots 1\ n)\ k\ \ldots n\text{-}1} \\
\mathbf{(n\text{-}1\ k\text{-}1\ \ldots 1)\ n\ k\ \ldots n\text{-}2} \\
\mathbf{(1\ \ldots k\text{-}1\ n\text{-}1)\ n\ k\ \ldots n\text{-}2} \\
\vdots \\
\mathbf{(k\ 1\ \ldots k\text{-}1)\ k{+}1\ \ldots n}
\end{array}$$

Informally, in employing this sequence of moves we essentially fix elements **1 ... k-1** and move the remaining element "around" them. Since $n-k$ is *even* and the order of σ is 2 (i.e. σ^2 leaves the puzzle unchanged) it follows that **1 ... k-1** will be in the correct numerical ordering after this sequence of moves.
Using this $k-cycle$ we will now exhibit a $3-cycle$. We shall apply the following moves to the puzzle:

$$\sigma\tau^{-1}\sigma\tau$$

Then we apply the inverse of our $k-cycle$ twice. Written linearly this is:

$$\begin{array}{c}
\mathbf{(1\ \ldots k)\ k{+}1\ \ldots n} \\
\mathbf{(k\ \ldots 1)\ k{+}1\ \ldots n} \\
\mathbf{(k\text{-}1\ \ldots 1\ k{+}1)\ k{+}2\ \ldots n\ k} \\
\mathbf{(k{+}1\ 1\ \ldots k\text{-}1)\ k{+}2\ \ldots n\ k} \\
\mathbf{(k\ k{+}1\ 1\ \ldots k\text{-}2)\ k\text{-}1\ k{+}2\ \ldots n} \\
\mathbf{(k{+}1\ 1\ \ldots k\text{-}2\ k)\ k\text{-}1\ k{+}2\ \ldots n} \\
\mathbf{(1\ \ldots k\text{-}2\ k\ k{+}1)\ k\text{-}1\ k{+}2\ \ldots n}
\end{array}$$

which yields the *consecutive 3-cycle* **(k k+1 k-1)**.

Now we shall apply the following lemma's (Lemma 1 is borrowed from [2]).

Lemma 1: *For $n \geq 3$, the consecutive 3-cycles generate A_n.*

Lemma 2: *Given A_n and an odd permutation, we can generate all of S_n.*

Proof:
Since σ is an odd permutation, if we take $\pi \in A_n$ then

$$\sigma \circ \pi \text{ and } \pi \circ \sigma$$

are odd permutations. Therefore these permutations are not in A_n. This implies that any other subgroup we generate will be *larger* than A_n. By Lagrange's Theorem [4, pg. 156] we know that the order of this subgroup must divide the order of the group. Therefore since $|A_n|$ is the largest divisor of $|S_n|$ (other than $|S_n|$), the *larger* subgroup generated is S_n. ✓

Now that we have generated A_n by Lemma 1, it remains to show that we have an odd permutation. By Lemma 2, the composition of this odd permutation with elements of A_n will generate S_n and confirm the puzzle is solvable. The odd permutation we seek is σ as $k \equiv 3 \pmod 4$ and thus $\lfloor \frac{k}{2} \rfloor$ is odd. Therefore the game is solvable. ✓

Theorem 3: *If $n \equiv 1 \pmod 2$ and $k \equiv 2 \pmod 4$, then the puzzle is solvable.*

Proof: We shall proceed in a manner similar to that of Theorem 2. We begin by mentally "glueing" the first two tiles together and moving them both out of the window. This allows us to take tiles **3 . . . k+1** and view them as *fixed* (i.e. never leaving) in the *swapping window*. Similar to the proof of Theorem 2 we now proceed to move the remaining tiles past the fixed elements in a series of $n-k+1$ counterclockwise swap-translations (i.e. $(\sigma\tau^{-1})^{n-k+1}$). Since $n-k+1$ is even we know that after this sequence of swap-translations that the fixed elements will remain in the correct counting order.

To better illustrate this we give the linear representation:

1 2 (3 . . . k+1 k+2) k+3 . . . n
1 2 (k+2 k+1 . . . 3) k+3 . . . n
1 2 k+2 (k+1 . . . 3 k+3) k+4 . . . n
1 2 k+2 (k+3 3 . . . k+1) k+4 . . . n
1 2 k+2 k+3 (3 . . . k+1 k+4) k+5 . . . n
⋮

We continue this a total of $n-k+1$ times to get:

(3 4 . . . k+1 1 2) k+2 . . . n

This allows us to see that we now have the pair to which we can apply the same process (i.e. begin by mentally "glueing" them together). We repeat this process of pairing elements of the swapping window $\frac{k}{2}$ times, at which point we will get the following configuration:

(k+1 1 2 . . . k-1) k k+2 . . . n

Which yields the $(k+1)-$*cycle* **(k+1 1 . . . k)**.
Using this $(k+1)-$*cycle* we shall now exhibit a *consecutive 3-cycle* to generate A_n. We

begin by performing a counterclockwise swap-translate, then a swap, and then apply the inverse our $(k+1) - cycle$ twice. Written linearly this is:

$$\begin{array}{c}
\mathbf{(1\ldots k)\ k{+}1\ldots n} \\
\mathbf{(k\ldots 1)\ k{+}1\ldots n} \\
\mathbf{k\ (k{-}1\ldots 1\ k{+}1)\ k{+}2\ldots n} \\
\mathbf{k\ (k{+}1\ 1\ldots k{-}1)\ k{+}2\ldots n} \\
\mathbf{k\ (1\ldots k{-}1\ k{+}2)\ k{+}1\ldots n} \\
\mathbf{(1\ldots k{-}1\ k{+}2)\ k\ k{+}1\ldots n}
\end{array}$$

Which yields the *consecutive 3-cycle* **(k+2 k k+1)**. Again using Lemma 1 we can now generate the alternating group, A_n. Since σ is an odd permutation (as $\frac{k}{2}$ is odd), we can now generate all of S_n by Lemma 2. Therefore the puzzle is solvable.✓

Theorem 4: *If $n \equiv 1$ (mod 2) and $k \equiv 0$ (mod 4) or $k \equiv 1$ (mod 4), then the puzzle is not solvable.*

Proof:
We first note that the proofs of Theorem 2 and Theorem 3 give us the same consecutive 3-cycles for these two given cases. Thus, in these two cases, we can generate A_n. However, also note that since σ is our only generator of the group in this case and $\lfloor \frac{k}{2} \rfloor$ is even, we have that σ is even and thus we do not have an odd permutation. It follows that any permutation π characterizing the configuration of a given puzzle can only be $\pi \in A_n$ and therefore the puzzle is unsolvable. ✓

Theorem 5: *If $n \equiv 0$ (mod 2) and $k \equiv 1$ (mod 4) or $k \equiv 3$ (mod 4), then the puzzle is not solvable.*

Proof:
Begin by coloring tiles alternately red and green. Note that the tiles are now partitioned by color and parity, that is to say, the red tiles form the set of even numbered positions in $\{1\ldots n\}$ and green tiles form the set of odd numbered positions. Since n is even it follows that there will be pairs of oppositely colored tiles. Furthermore, since k is odd any swap will result in an element remaining fixed (i.e. the center tile of the *swapping window*).

Next, enumerate the *positions* of tiles beginning with 1 on the game, again using the left-most position of the *swapping window* as a reference point. Since every swap fixes a center point, the only permutations generated transpose tiles of the same parity and therefore the same color. Thus, if two adjacent tiles are of the same color, we cannot create any permutation that will transpose the two. Therefore the puzzle is not solvable.✓

2 Proof of Bounds

We begin by explaining why an asymptotic bound is worth examining as opposed to a fixed n and k. Note that k is always bounded above by n. This is simply due to the construction of the puzzle itself. This doesn't mean that k couldn't necessarily grow infinitely as n does, as long as $k < n$. However, we also need some constraints on how much bigger n needs to be than k. Because it is unclear how k should be allowed to grow as n grows infinitely, it makes sense to simply consider k as some fixed constant and leave n unbounded.

Another issue that we want to explain is why we only consider swaps when calculating the time complexity of solving a given puzzle. Recall that a swap is the operation defined by the reflection of the k tiles in the swapping window and a translation is the movement defined by sliding tiles within the track (see section 1 for more details). There are two reasons why we only consider swaps when calculating the time complexity. Firstly, we note that translations are not used when considering the lower bound of the time complexity. Secondly, translations only add a constant multiplicative factor to the upper bound of the time complexity. To see why this second fact is true, consider fixing the leftmost position of the swap window to be the leftmost position of the puzzle. In this case, we have that performing a translation of however many positions is equivalent to changing the value of a pointer. In other words, any translation is $O(1)$ and not $O(x)$ for a translation of x positions. It follows that translations do not need to be considered in the calculation of the runtime complexity. Swaps, on the other hand, involve the transposition of elements ($\lfloor \frac{k}{2} \rfloor$ such transpositions in fact). In this respect one can see that a swap is the only real elementary puzzle operation that would require computation for a given representation of the puzzle. The author also notes that upon a successful implementation of the puzzle in the Java programming language, this fact held true.

2.1 Upper Bound

We will break the proof of this upper bound into two theorems. The first will prove an upper bound in the case of $n \equiv 0 \ (mod\ 2)$ and $k \equiv 0 \ (mod\ 2)$. The second theorem will prove an upper bound in the two other solvable cases of $n \equiv 1 \ (mod\ 2)$ and $k \equiv 3 \ (mod\ 4)$ as well as $n \equiv 1 \ (mod\ 2)$ and $k \equiv 2 \ (mod\ 4)$. All of the cases examined in this proof are solvable by Theorems 1, 2, and 3 in section 1. These theorems in section 1 use certain algorithms to prove solvability and said algorithms only apply to certain solvable cases, which is why we separate these cases into two theorems here. We begin by defining an abstract puzzle operation that will help us in the proof of both these theorems:

Definition: *A swapping cycle γ_x is the act of moving the x^{th} tile (i.e. the tile with the number x on it) into the rightmost position of the swap window via clockwise translations, performing a swap, and repeating. For example, if we let $x = 1$, without loss of generality, it*

involves the following series of puzzle configurations. Note that the numbers in parentheses represent the swap window, underlined terms represent a single tile, and that the starting configuration is the identity (or "solved") configuration for display purposes only:

$$\Big(\underline{1}\ \underline{2}\dots\ \underline{k}\Big)\ \underline{k+1}\dots\ \underline{n}$$
$$\downarrow$$
$$\Big(\underline{n-k+2}\ \underline{n-k+3}\ \dots\ \underline{n-1}\ \underline{n}\ \underline{1}\Big)\ \underline{2}\ \dots\ \underline{n-k+1}$$
$$\downarrow$$
$$\Big(\underline{1}\ \underline{n}\ \underline{n-1}\ \dots\ \underline{n-k+3}\ \underline{n-k+2}\Big)\ \underline{2}\ \dots\ \underline{n-k+1}$$
$$\downarrow$$
$$\Big(\underline{n-2k+3}\ \underline{n-2k+4}\ \dots\ \underline{1}\Big)\ \underline{n}\ \underline{n-1}\ \dots\ \underline{n-k+3}\ \underline{n-k+2}\ \underline{2}\ \dots\ \underline{n-2k+2}$$

Now that we have defined γ_x, we can go ahead and prove the following theorem.

Theorem 6: *There exists an algorithm A_{TS} that solves any configuration of the puzzle in the case of $n \equiv 0 \pmod 2$ and $k \equiv 0 \pmod 2$ such that the runtime of A_{TS}, denoted as $T(A_{TS})$, is such that $T(A_{TS}) \in O(n^2)$.*

Proof:

We will begin by defining A_{TS} and proving its correctness (i.e. that it solves any configuration of the puzzle in the given case). We will then analyze its running time $T(A_{TS})$ and prove that $T(A_{TS}) \in O(n^2)$.

A_{TS} can be defined by a two step process that is repeated for each tile in the puzzle. Let $i = 2, \dots, n$ be the i^{th} tile (i.e. the tile with the number i on it) and let ϕ be the permutation of the tiles given by the most recent configuration of the puzzle at the beginning of step (1) (note that ϕ will change upon every completion of the two steps). We are ignoring $i = 1$ here because we will fix $\phi(1) = 1$ for any ϕ. We can now view the puzzle in a linear fashion, fixing 1 in the leftmost position. For each given i, do the following in order, starting from $i = 2$ (see Figure 2.1 for an example):

(1) Move the given tile to within $k - 1$ of where it should be in the identity configuration ("solved" configuration). More specifically, if the configuration of the puzzle after we apply the first step is defined by the permutation π, and the number on the tile is i, then we want $|\pi(i) - i| < k - 1$. We accomplish this by applying $\lfloor \frac{|\phi(i)-i|}{k-1} \rfloor$ applications of γ_i. We are assuming that we have moved the i^{th} tile into the rightmost position of the swap window at the beginning of each step for each tile. In other words, the value $\lfloor \frac{|\phi(i)-i|}{k-1} \rfloor$ is such that we

make ϕ be defined as having the i^{th} tile in the rightmost position of the swap window. Also note that if $|\phi(i) - i| < k - 1$, then this is equivalent to doing nothing in this step.

(2) Apply $|\pi(i) - i|$ adjacent transpositions ω to the given tile in order to move it to its correct position in the identity configuration (where π is as it is defined above). We know such an adjacent transposition can be generated via the proof of Theorem 1 in section 1. By applying these adjacent transpositions, we are resolving the remaining distance between the current tile and its correct place in the solved puzzle.

$$\underline{1}\ \underline{4}\ \underline{6}\ \underline{7}\ \underline{10}\ \underline{3}\ \left(\underline{5}\ \underline{9}\ \underline{8}\ \underline{2}\right)\ \underline{12}\ \underline{11}$$
$$\downarrow_{\gamma_2}$$
$$\underline{1}\ \underline{4}\ \underline{6}\ \underline{7}\ \underline{10}\ \underline{3}\ \left(\underline{2}\ \underline{8}\ \underline{9}\ \underline{5}\right)\ \underline{12}\ \underline{11}$$
$$\downarrow_{\gamma_2}$$
$$\underline{1}\ \underline{4}\ \underline{6}\ \left(\underline{7}\ \underline{10}\ \underline{3}\ \underline{2}\right)\ \underline{8}\ \underline{9}\ \underline{5}\ \underline{12}\ \underline{11}$$
$$\downarrow_{\gamma_2}$$
$$\underline{1}\ \underline{4}\ \underline{6}\ \left(\underline{2}\ \underline{3}\ \underline{10}\ \underline{7}\right)\ \underline{8}\ \underline{9}\ \underline{5}\ \underline{12}\ \underline{11}$$
$$\downarrow_{\omega}$$
$$\underline{1}\ \underline{4}\ \underline{2}\ \left(\underline{6}\ \underline{3}\ \underline{10}\ \underline{7}\right)\ \underline{8}\ \underline{9}\ \underline{5}\ \underline{12}\ \underline{11}$$
$$\downarrow_{\omega}$$
$$\underline{1}\ \underline{2}\ \underline{4}\ \left(\underline{6}\ \underline{3}\ \underline{10}\ \underline{7}\right)\ \underline{8}\ \underline{9}\ \underline{5}\ \underline{12}\ \underline{11}$$

Figure 2: An example of steps 1 and 2 of A_{TS} for $i = 2$ on a puzzle where $n = 12$, $k = 4$ (the numbers in parentheses represent the elements in the swap window).

To prove that this algorithm is correct, we first note that in order to solve any configuration in this case, it suffices to move tiles that are to the right of their correct position to the left into their correct position (assuming they stay there). Consequently, we need to show that the following two invariants hold for each tile in the puzzle during the application of the algorithm. The first invariant is that after both steps of the algorithm, the given tile is in it's correct position. The second invariant is that once a tile is in its correct position, it stays there for the remainder of the algorithm. If we can prove that both of these invariants hold, it follows that all tiles are in their correct position and stay that way for the entire algorithm and consequently the puzzle is solved upon termination of A_{TS}.

To prove the first invariant, we need to show that if $x_i = |\phi(i) - i|$ then the i^{th} tile is moved x_i positions to the left into its correct position. To prove this, we first note that $x_i = p \cdot (k-1) + r$ for some $p, r \in \mathbb{N}$. It follows that $p = \lfloor \frac{|\phi(i)-i|}{k-1} \rfloor$ (i.e. the amount of spaces the i^{th} tile is moved during step 1 of A_{TS}) and consequently $r = |\pi(i) - i|$ where π is as it's defined in step 1 of A_{TS}. Thus, we have that for each $i = 2, \ldots, n$, the i^{th} tile moves x_i spaces to the left upon completion of steps 1 and 2 of A_{TS} and consequently is placed into the position it would be in in the identity configuration.

To prove the second invariant, all we need to show is that an application of γ_x as it is described in step 1 of A_{TS} and an application of an adjacent transposition ω as it is described in step 2 of A_{TS} only moves the given tile and leaves all other tiles to the left of it in the identity configuration fixed. This is obviously true of ω merely by its definition. To see why this is true for γ_x, note that we are only using γ_x in step 1 of A_{TS} up until it is within $k-1$ of where it should be in the identity. Because we are performing the two steps of A_{TS} iteratively, in order (i.e. from 2 to n), we have that by getting to within $k-1$ of where it should be in the identity, applications of γ_x will not disrupt any tiles to the left of position x and thus the invariant holds.

All that is left is to analyze the runtime $T(A_{TS})$ of A_{TS} and prove that $T(A_{TS}) \in O(n^2)$. We begin by noting that a possible worst case configuration of this algorithm would be if the puzzle were in reverse order (i.e. start from n and going down to 1, clockwise) because this configuration would have the most elements in the wrong order. So, examining this case is sufficient for this analysis and we leave it to the reader to see that an analysis of this case would apply to all other cases with essentially the same results.

Let $f(n, k, i)$ be the number of operations needed to perform γ_{i+1} on a puzzle with n tiles and a swapping window of size k. Since γ_{i+1} is used at most $\lfloor \frac{n-1-i}{k-1} \rfloor$ many times in any iteration of step (1) of A_{TS}, we have that the time complexity of any iteration of step (1) of A_{TS} would be $(\lfloor \frac{n-1-i}{k-1} \rfloor) \cdot f(n, k, i)$. Next, let $g(n, k)$ be the number of operations needed to perform an adjacent transposition ω. We have that the remaining distance that needs to be resolved between any given tile and its correct position would be at most $k-2$ at the beginning of any iteration of step (2) of A_{TS}. Thus, it would follow that at most $k-2$ many adjacent transpositions would need to be made in any iteration of step (2) of A_{TS} and thus the time complexity of any iteration of step (2) of A_{TS} would be $(k-2) \cdot g(n, k)$. So, since we run step (1) and (2) of A_{TS} $n-1$ times (for the tiles $i = 2, \ldots, n$), it follows that the overall runtime can be described with the following equation:

$$T(A_{TS}) = \sum_{i=1}^{n-1} \left(\left\lfloor \frac{n-1-i}{k-1} \right\rfloor \right) \cdot f(n, k, i) + (k-2) \cdot g(n, k)$$

A quick analysis of γ_{i+1} gives us that $f(n, k, i) = 1$ because we are ignoring all translations

and so we only care about the one swap performed. Also, by analyzing the adjacent transposition algorithm given in the proof of Theorem 1 in section 1 (keeping in mind to ignore translations), we have that $g(n,k) = n+2$. So, we have that the summation above simplifies to:

$$T(A_{TS}) = \sum_{i=1}^{n-1} \left(\left\lfloor \frac{n-1-i}{k-1} \right\rfloor \right) \cdot 1 + (k-2) \cdot (n+2) = c_1 \cdot n + c_2 \cdot n^2$$

for some $c_1, c_2 \in \mathbb{N}$, which gives us that $T(A_{TS}) \in O(n^2)$. ✓

Theorem 7: *There exists an algorithm B_{TS} that solves any configuration of the puzzle in the cases of $n \equiv 1 \pmod 2$ and $k \equiv 3 \pmod 4$ as well as $n \equiv 1 \pmod 2$ and $k \equiv 2 \pmod 4$ such that the runtime of B_{TS}, denoted as $T(B_{TS})$, is such that $T(B_{TS}) \in O(n^2)$.*

Proof:

We will go about defining B_{TS} in almost the exact same way we defined A_{TS} in the proof of Theorem 6. However, because we do not have an explicit algorithm for an adjacent transposition in these cases, we will resort to using the consecutive 3-cycles defined in section 1 for these solvable cases. Recall that a consecutive 3-cycle is a cycle of size 3 equal to $(k\ k+1\ k-1)$ for some k. More specifically, we will use said 3-cycles to generate specific even permutations. Before we define B_{TS}, let us discuss some group-theoretic concepts that will make the process of defining B_{TS} easier. First, consider the following lemma:

Lemma 1: Any permutation $\pi \in S_n$ can be factored into the composition of two permutations α and β (i.e. $\pi = \alpha \circ \beta$) such that $\alpha \in A_n$ and $\beta \in (S_n - A_n) \cup \{e\}$ or in other words where α is an even permutation and β is an odd permutation or β is the identity permutation (equivalent to performing no operations on the puzzle).

Proof:

We know that any permutation π is the product of t transpositions $\tau_1\tau_2\ldots\tau_t$ and that the parity of π is even if t is even and odd if t is odd. Thus, if we have that π is even, we just let $\alpha = \pi$ and $\beta = e$ and if π is odd, we let $\alpha = e$ and $\beta = \pi$. ✓

We can take Lemma 1 one step further and note that β can be any arbitrary odd permutation if we allow α to vary (i.e. $\alpha = \pi \circ \beta^{-1}$ for any $\beta \in (S_n - A_n) \cup \{e\}$). Next, consider the permutation defined by a swap σ. Let $(a_1, a_2, \ldots, a_k)$ be the configuration of the k tiles inside the swapping window with a_1 in position 1. For the first case of $k \equiv 3\ (mod\ 4)$ we have that $\sigma = (a_1a_k)(a_2a_{k-1})\ldots(a_{\lceil \frac{k}{2} \rceil + 1}a_{\lceil \frac{k}{2} \rceil - 1})$ where the center element $a_{\lceil \frac{k}{2} \rceil}$ stays fixed. For $k \equiv 2\ (mod\ 4)$, we have that $\sigma = (a_1a_k)(a_2a_{k-1})\ldots(a_{\frac{k}{2}}a_{\frac{k}{2}+1})$. Given this, we have that

the odd permutation β from Lemma 1 can simply be a swap in the two given cases because $k \equiv 3 \ (mod\ 4)$ and $k \equiv 2 \ (mod\ 4)$ are both such that $\lfloor \frac{k}{2} \rfloor$ is odd (i.e. an odd number of transpositions occur during each swap). Given this information, if we let π be the permutation that defines the starting configuration of the puzzle and let $\alpha = \pi \circ \beta^{-1}$ where β is either a single swap or the identity operation, we have that performing the necessary puzzle operations equivalent to the permutation α^{-1} on the puzzle will solve it.

Now we can define B_{TS} as follows. It will be the same two step process done on each tile in A_{TS} except with some changes to each step. The first change is in step 1, and it is for us to apply γ_i to get the i^{th} tile to be within $2k-1$ (i.e. $< 2k-1$) of where it should be in the identity as opposed to $k-1$ in A_{TS}. This will help us when we implement the next change, which is a new invariant that must hold. If α is the current permutation of the puzzle and $\overline{\alpha}$ is the permutation defined by applying the first step to the given tile, we must have that both α and $\overline{\alpha}$ are even. In order to make sure that this invariant is enforced right from the start, we must do the following. Let π be the starting configuration of the puzzle. If $\pi \in A_n$, we do nothing to the puzzle and if $\pi \in S_n - A_n$, we perform a single swap as was discussed above. Now we have that it is possible for the invariant to hold from the beginning.

In order for the invariant to hold throughout B_{TS}, we need to make sure that moving the i^{th} tile to within $2k-1$ of where it needs to be results in an even permutation. This can be accomplished by doing the following. Note that every application of γ_i will flip the parity of the current permutation of the puzzle as we are performing one swap per application of γ_i and this swap is an odd permutation by the argument made above. Thus, if we need to do an even number of such applications of γ_i then we have that the new invariant holds and we are good. However, if we were originally supposed to perform an odd number of applications of γ_i, we can now perform one less γ_i which will result in an overall even number of swaps and consequently an even permutation. We are able to avoid performing this extra γ_i because of the change from getting the i^{th} tile to be within $2k-1$ as opposed to $k-1$ (i.e. there is more distance to resolve between the i^{th} tile and its correct location). Thus we have that we can ensure that the new invariant holds for each application of step 1 on each tile.

This new invariant helps set us up for our third and final change, which takes place in step 2. As previously mentioned, we do not have an explicit algorithm for an adjacent transposition for the given cases. We do, however, have explicit algorithms for consecutive 3-cycles in the given cases. We are going to use these 3-cycles to resolve the remaining distance between the i^{th} tile after step 1 and where it should be in the identity. We will do this by applying the inverse of the permutation α_i that characterizes the $2k-1$ tiles within the i^{th} position. We have that α_i will be even because of our new invariant. So, it just comes down to generating α_i in order to resolve this remaining difference. We can do this by showing that we can generate all of A_m with just consecutive 3-cycles (where $m = 2k-1$ in this case). More specifically, we want to show that it only takes a quadratic number in m of such consecutive

3-cycles to generate A_m. To show this, we reference the following lemma from [2]. We will observe that the proof of this lemma as it is given in [2] allows us to conclude that the number of consecutive 3-cycles used is quadratic in m:

Lemma 2: For $m \geq 3$, any $\alpha \in A_m$ is the product of $O(m^2)$ consecutive 3-cycles.

Given this lemma, and the consecutive 3-cycle algorithm presented in section 1, we have that α_i can be generated in these cases. If we now repeat these two steps for each $i = 2, \ldots, n$ just as we did in A_{TS}, we will have that the algorithm will solve the puzzle.

The proof of the correctness of B_{TS} is exactly the same as it was for A_{TS} except that we make note of the changes made. The change from $k-1$ to $2k-1$ doesn't change the correctness argument made in A_{TS} and because we already have that the new invariant holds, we have that the algorithm is correct. As for the analysis of B_{TS}, let us consider the same case we did for A_{TS} of the reversed puzzle, noting once again that the reader can conclude that any other case analysis will result in essentially the same results.

Let $f(n,k,i)$ be the same as it was in the analysis of A_{TS}. Since γ_{i+1} is used at most $\lfloor \frac{n-1-i}{2k-1} \rfloor$ many times in any iteration of step (1) of B_{TS}, we have that the time complexity of any iteration of step (1) of B_{TS} would be $(\lfloor \frac{n-1-i}{2k-1} \rfloor) \cdot f(n,k,i)$. Next, let $h(n,k,i)$ be the cost of generating α_{i+1} on a puzzle with n tiles and k elements in the swapping window. Since α_{i+1} is generated once per iteration of step (2) of B_{TS}, it follows that the time complexity of step (2) of any iteration of B_{TS} is just $h(n,k,i)$. So, if $T(B_{TS})$ is the running time of B_{TS}, then:

$$T(B_{TS}) = \sum_{i=1}^{n-1} \left(\left\lfloor \frac{n-1-i}{2k-1} \right\rfloor \right) \cdot f(n,k,i) + h(n,k,i)$$

A quick analysis of the consecutive 3-cycle algorithms presented in section 1 shows that $h(n,k,i) = c \cdot m^2 \cdot n = c' \cdot n$ where $c, c' \in \mathbb{N}$. This is because $m = 2k-1$ which is essentially a constant by assumption. So, we have that

$$T(B_{TS}) = \sum_{i=1}^{n-1} \left(\left\lfloor \frac{n-1-i}{2k-1} \right\rfloor \right) \cdot f(n,k,i) + h(n,k,i) = c_1 \cdot n + c_2 \cdot n^2 \in O(n^2)$$

for some $c_1, c_2 \in \mathbb{N}$, which gives us that $T(B_{TS}) \in O(n^2)$. ✓

Theorems 6 and 7 allow us to generalize our upper bound to all solvable puzzles, as stated in the following theorem:

Theorem 8: *Any puzzle with n tiles and a swapping window of size k such that n and k yield a solvable case can be solved using $O(n^2)$ swaps.*

2.2 Lower Bound

We begin by defining a property of a permutation $\pi \in S_n$:

Definition: *A triple inversion is a property of a permutation $\pi \in S_n$ such that for $i < j < k$ for $i, j, k \in \{1, \ldots, n\}$, we have that if we cyclically fix $\pi(i)$ as our first element and continue examining π in a clockwise manner, we will reach $\pi(k)$ before we reach $\pi(j)$.*

The maximum number of triple inversions any permutation can have is equal to $\binom{n}{3}$. Thus, any solution to the puzzle would, in the worst case, have to employ a number of swaps such that all $\binom{n}{3}$ triple inversions are resolved. All that is left to do is show how many triple inversions can be resolved by a swap. We start by proving the following theorem:

Theorem 9: *Leaving everything equal up to translation, a swap can only resolve at most $c \cdot n$ triple inversions at a time for some $c \in \mathbb{R}$.*

Proof:

Let $T = \{t_1, t_2, t_3\}$ be a triple inversion and let $S = \{s_1, \ldots, s_k\}$ be the elements within the given swap window configuration. Because any triple inversion is made up of three elements, any combination of which can be in the swap window at any time, we can break down the number of triple inversions that a single swap can resolve into the following four cases:

(1) $|T \cap S| = 0$
In this case, a single swap won't resolve T and no such triple inversions are resolved.

(2) $|T \cap S| = 1$
In this case, a single swap won't resolve T and no such triple inversions are resolved.

(3) $|T \cap S| = 2$
This case has two sub cases. The first is when the two end points of the triple inversion are in the swap window (i.e. $T \cap S = \{t_1, t_3\}$). In this case, all such inversions are resolved. Since both of the endpoints must lie in the swap window and the third element is outside of the swap window, we have that there are $\binom{k}{2}$ such inversions. The second sub case is when one end point and the middle point are in the swap window (i.e. $T \cap S = \{t_1, t_2\}$ or $T = \{t_2, t_3\}$). For $T = \{t_1, t_2\}$, we have $\binom{k}{2}$ possibilities for the tiles in the swap window, but this time we need to account for the third element of the triple that is not in the swap window. So, since the same holds for when $T = \{t_2, t_3\}$, the total number of triple inversions that are resolved in this sub case is $2(n-k) \cdot \binom{k}{2}$. So, in the whole case of $|T \cap S| = 2$, we have that the

maximum number of triple inversions that could be resolved is $\binom{k}{2} + 2(n-k) \cdot \binom{k}{2}$.

(4) $|T \cap S| = 3$
In this case, T will be resolved because the ordering of t_1, t_2, and t_3 will be reversed. So, there are at most $\binom{k}{3}$ such triple inversions in any given configuration.

By combining all three cases, we get that the maximum number of triple inversions resolved by any single swap is $\binom{k}{2} + 2(n-k) \cdot \binom{k}{2} + \binom{k}{3} = c \cdot n$ for some $c \in \mathbb{R}$. ✓

Theorem 10: *Any algorithm used to solve the puzzle must use $\Omega(n^2)$ swaps.*

Proof: Because any swap can only resolve at most $c \cdot n$ triple inversions as proved in Theorem 9, and because any puzzle configuration can have at most $\binom{n}{3}$ triple inversions, we have that we need to perform $\frac{\binom{n}{3}}{c \cdot n} = \frac{n(n-1)(n-2)}{6 \cdot c \cdot n} = \frac{(n-1)(n-2)}{c'} \in \Omega(n^2)$ operations, where $c, c' \in \mathbb{R}$. ✓

We can now state our final theorem regarding a tight asymptotic bound on the number of swaps needed to solve any solvable puzzle.

Theorem 11: *Any puzzle with n tiles and k elements in the swapping window such that n grows infinitely, k stays fixed, and n and k provide one of the solvable cases as stated in section 1 requires $\Theta(n^2)$ swaps to be solved.*

Proof: This simply follows from Theorem 8 and Theorem 10, and the definition of Θ bounds. ✓

3 Conclusion and References

We have just proved a tight asymptotic bound of n^2 for the number of swaps required to solve a puzzle of n tiles and a swapping window of size k such that we let n grow infinitely and fix k. There are a few things that weren't touched upon in the various proofs of this paper that are worth mentioning, however. The first is that even though we considered k fixed and n unbounded, because all analyses were essentially in closed form, an analysis of an unbounded k that was only bounded above by n and still allowed for a solvable puzzle is possible. The other thing worth mentioning is that an analysis of some of the unsolvable cases is also possible. We first note that there were two unsolvable cases presented in section 1. The first of these cases included when $n \equiv 1 \ (mod\ 2)$ and $k \equiv 0 \ (mod\ 4)$ as well as $n \equiv 1 \ (mod\ 2)$ and $k \equiv 1 \ (mod\ 4)$. In these cases, we get that A_n can be generated, but not S_n. Given this information, we can see that using the same rationale used in the proof of Theorem 7, and the fact that the lower bound argument still applies to these cases, we get

that the following remark is true:

Remark: The unsolvable cases of the puzzle where we have that all permutations of the puzzle are in A_n but not $S_n \setminus A_n$ can have all solvable configurations (i.e. those whose characterizing permutation live in A_n) solved in $\Theta(n^2)$ swaps.

The second unsolvable case is the one in which $n \equiv 0 \ (mod\ 2)$ and $k \equiv 1 \ (mod\ 2)$. Because the subgroup of S_n generated by the swaps and translations of this particular puzzle cannot be determined, we have no way of knowing whether or not arbitrary permutations of the tiles in this case of the puzzle can be generated in $O(n^2)$ swaps. This case is therefore left as an open problem.

4 Appendix

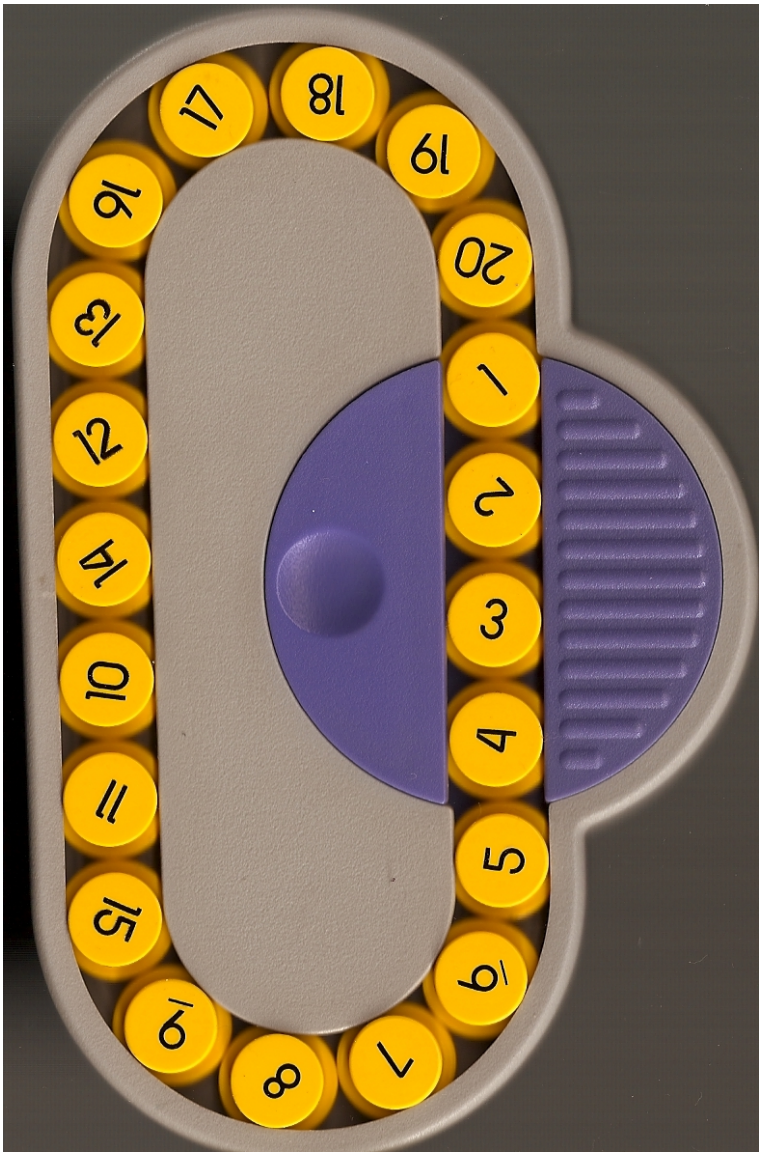

Figure 3: Top Spin puzzle with 20 tiles and a swapping window of size 4.

References

[1] S. Kaufmann, A. Kavountzis, *Proof of Solvability for the Generalized Oval Track Puzzle*, Rose-Hulman Undergraduate Math Journal; Vol. 10, Issue 1, 2009.

[2] A. F. Archer, *A Modern Treatment of the 15 Puzzle*, American Mathematical Monthly, November 1999, pp. 793-799.

[3] E. Wilbur, *Topspin: Solvability of Sliding Number Games*, Rose-Hulman Undergraduate Math Journal; Vol. 2, Issue 2, 2001.

[4] J.J. Rotman, *A First Course in Abstract Algebra*, Prentice Hall, 2006.